

Lecture 5: Working with Data

Reviewing what is a data set

- A data set is a collection of logically related data records stored on one disk storage volume or a set of volumes.
- A data set can be:
 - a source program
 - a library of macros
 - a file of data records used by a processing program.
- You can print a data set or display it on a terminal. The logical record is the basic unit of information used by a program running on z/OS.

Types of Data Sets: Sequential, Partitioned & VSAM

- A **sequential data set** is a collection of records written and read in sequential order from beginning to end
- A **partitioned data set (PDS)** is a collection of sequential data sets, called members.
 - Consists of a directory and one or more *members*
 - The *members* are sequential data sets
 - Also called a *library*
- A **VSAM data set** (*Virtual Storage Access Method*) provides more advanced capabilities in terms of storage and access

Key Data Set Characteristics

- Two important characteristics of data files are
 - DATA ORGANIZATION
 - METHOD OF ACCESS
- **Data organization** refers to the way the records of the file are organized on the backing storage device:
 - **Sequential** - Records organized serially.
 - **Relative** - Relative record number based organization.
 - **Indexed** - Index based organization.
- The **method of access** refers to the way in which records are accessed:
 - **Sequential** - Accessed in sequence order
 - **Random** - Specify a key to get the data record
 - **Dynamic** - Can access sequentially or randomly

Working with Data Sets

Data Set Records and Fields

- We use the term **FIELD** to describe an item of information we are recording about an object
(e.g. StudentName, DateOfBirth, CourseCode).
- We use the term **RECORD** to describe the collection of fields which record information about an object
(e.g. a StudentRecord is a collection of fields recording information about a student).
- It is important to distinguish between the record occurrence (i.e. the values of a record) and the record type (i.e. the structure of the record).

Every record in a dataset has a **different value** but the **same structure**.

Data Set Records and Fields

STUDENTS.DAT

StudId	StudName	DateOfBirth
9723456	COUGHLAN	10091961
9724567	RYAN	31121976
9534118	COFFEY	23061964
9423458	O'BRIEN	03111979
9312876	SMITH	12121976

occurrences

```
DATA DIVISION.
```

```
FILE SECTION.
```

```
FD StudentFile.
```

```
01 StudentDetails.
```

```
02  StudId      PIC 9(7).
```

```
02  StudName    PIC X(8).
```

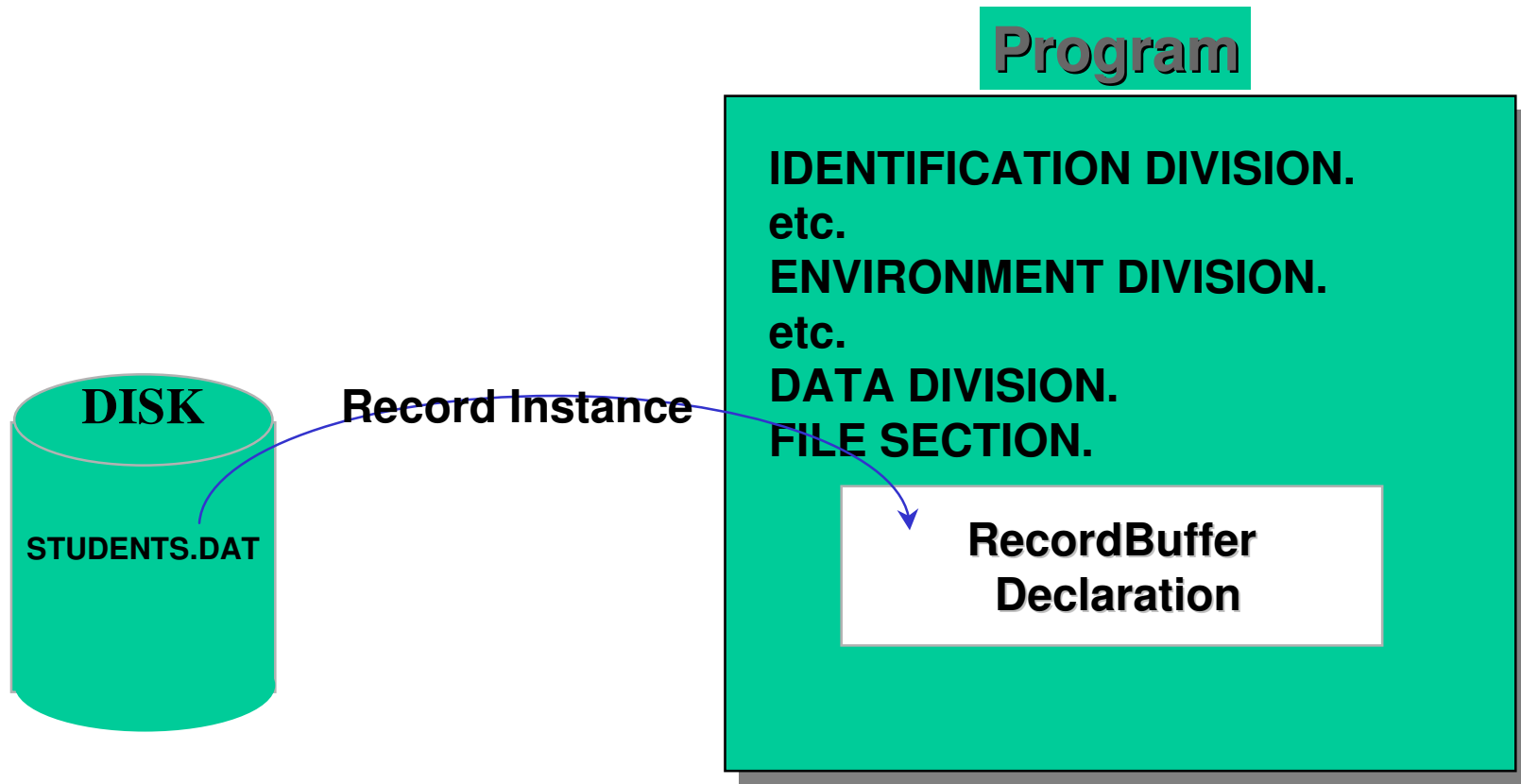
```
02  DateOfBirth PIC X(8).
```

Record Type
(Template)
(Structure)

Record Buffers

- To process a dataset, records are read from the file into the computer's memory **one record at a time**.
- The computer uses the programmers description of the record (i.e. the record template) to set aside sufficient memory to store **one instance** of the record.
- Memory allocated for storing a record is usually called a “**record buffer**”
- The record buffer is the only connection between the program and the records in the file.

Record Buffers



Implications of 'Buffers'

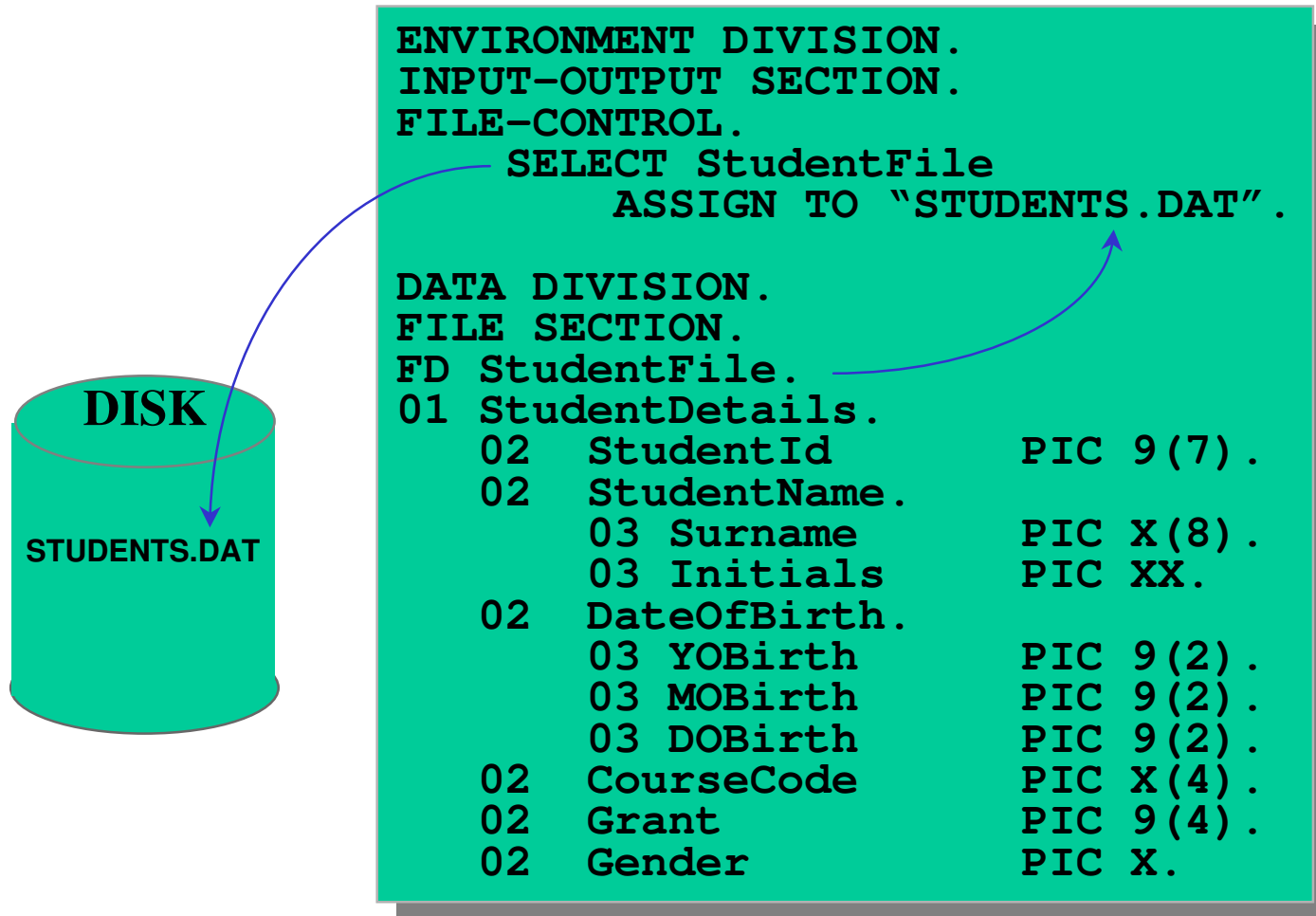
- If your program processes more than one dataset you will have to describe a record buffer for each dataset.
- To process all the records in an **INPUT file** each record instance must be copied (read) from the file into the record buffer when required.
- To create an **OUTPUT file** containing data records each record must be placed in the record buffer and then transferred (written) to the file.
- To transfer a record from an input file to an output file we will have to
 - read the record into the input record buffer
 - transfer it to the output record buffer
 - write the data to the output file from the output record buffer

Describing the record buffer in COBOL

```
DATA DIVISION.  
FILE SECTION.  
FD StudentFile.  
01 StudentDetails.  
    02 StudentId          PIC 9(7) .  
    02 StudentName.  
        03 Surname        PIC X(8) .  
        03 Initials       PIC XX .  
    02 DateOfBirth.  
        03 YOBirth        PIC 9(2) .  
        03 MOBirth        PIC 9(2) .  
        03 DOBirth        PIC 9(2) .  
    02 CourseCode        PIC X(4) .  
    02 Grant              PIC 9(4) .  
    02 Gender             PIC X .
```

- The record type/template/buffer of every file used in a program must be described in the FILE SECTION by means of an FD (file description) entry.
- The FD entry consists of the letters FD and an internal file name.

The Select and Assign Clause.



The internal file name used in the FD entry is connected to an external file (on disk or tape) by means of the Select and Assign clause.

The syntax of the FD statement

```
FD   file-name  
      [BLOCK CONTAINS integer-1 RECORDS]  
      [RECORD CONTAINS integer-2 CHARACTERS]
```

Select and FD statements for a non-VSAM file on an IBM mainframe

```
SELECT INVMAS  ASSIGN TO INVMAS  
      FILE STATUS IS INVMAS-FILE-STATUS.
```

```
FD   INVMAS  
    RECORD CONTAINS 70 CHARACTERS.
```

File Handling Verbs

COBOL file handling Verbs

- **OPEN**

Before your program can access the data in an input file or place data in an output file you must make the file available to the program by **OPEN**ing it.

- **READ**

The **READ** copies a record occurrence/instance from the file and places it in the record buffer.

- **WRITE**

The **WRITE** copies the record it finds in the record buffer to the file.

- **CLOSE**

Failure to close previously opened files may result in data not being written to the file or users being prevented from accessing the file.

The Open statement for sequential files

$$\underline{\text{OPEN}} \left\{ \begin{array}{l} \underline{\text{INPUT}} \text{ file-name-1} \dots \\ \underline{\text{OUTPUT}} \text{ file-name-2} \dots \\ \underline{\text{I-O}} \text{ file-name-3} \dots \\ \underline{\text{EXTEND}} \text{ file-name-4} \dots \end{array} \right\} \dots$$

Description

- You use the Open statement to open files in one of four modes.
- *Input mode* means the program will only read records from the file.
- *Output mode* means the program will create the file and can then write records to it.
- *I-O mode* means the program can read records from the file and can write them back to the file in the same locations.
- *Extend mode* means the program can add records to the end of an existing file.

The READ Ahead

- With the “read ahead” strategy we always try to stay one data item ahead of the processing.
- The general format of the “read ahead” algorithm is as follows;

```
Attempt to READ first data item
WHILE NOT EndOfStream
    Process data item
    Attempt to READ next data item
ENDWHILE
```

- Use this to process any stream of data.

Reading a Sequential File

- Algorithm Template

```
READ StudentRecords
  AT END MOVE HIGH-VALUES TO StudentRecord
END-READ
PERFORM UNTIL
  StudentRecord = HIGH-VALUES
DISPLAY StudentRecord
  READ StudentRecords
  AT END MOVE HIGH-VALUES
    TO StudentRecord
  END-READ
END-PERFORM
```

- This is an example of an algorithm which is capable of processing any sequential file; ordered or unordered!

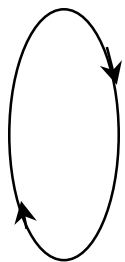
How the READ works

StudentRecord

StudentID	StudentName										Course.									
9334567	F	r	a	n	k		C	u	r	t	a	i	n			L	M	0	5	1

9334567	F	r	a	n	k		C	u	r	t	a	i	n			L	M	0	5	1
9383715	T	h	o	m	a	s		H	e	a	l	y				L	M	0	6	8
9347292	T	o	n	y		O		B	r	i	a	n				L	M	0	5	1
9378811	B	i	l	l	y		D	o	w	n	e	s				L	M	0	2	1

EOF

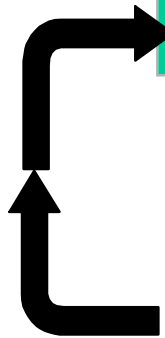


PERFORM UNTIL StudentRecord = HIGH-VALUES
READ StudentRecords
AT END MOVE HIGH-VALUES TO StudentRecord
END-READ
END-PERFORM.

How the READ works

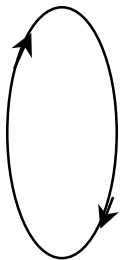
StudentRecord

StudentID	StudentName	Course.
9383715	ThomasHealy	LM068



9	3	3	4	5	6	7	F	r	a	n	k		C	u	r	t	a	i	n			L	M	0	5	1
9	3	8	3	7	1	5	T	h	o	m	a	s		H	e	a	l	y				L	M	0	6	8
9	3	4	7	2	9	2	T	o	n	y		O	'	B	r	i	a	n				L	M	0	5	1
9	3	7	8	8	1	1	B	i	l	l	y		D	o	w	n	e	s				L	M	0	2	1

EOF



PERFORM UNTIL StudentRecord = HIGH-VALUES

READ StudentRecords

AT END MOVE HIGH-VALUES TO StudentRecord

END-READ

END-PERFORM.

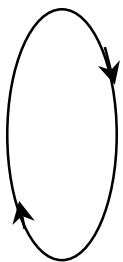
How the READ works

StudentRecord

StudentID	StudentName												Course.				
9347292	T	o	n	y	O	'	B	r	i	a	n		L	M	0	5	1

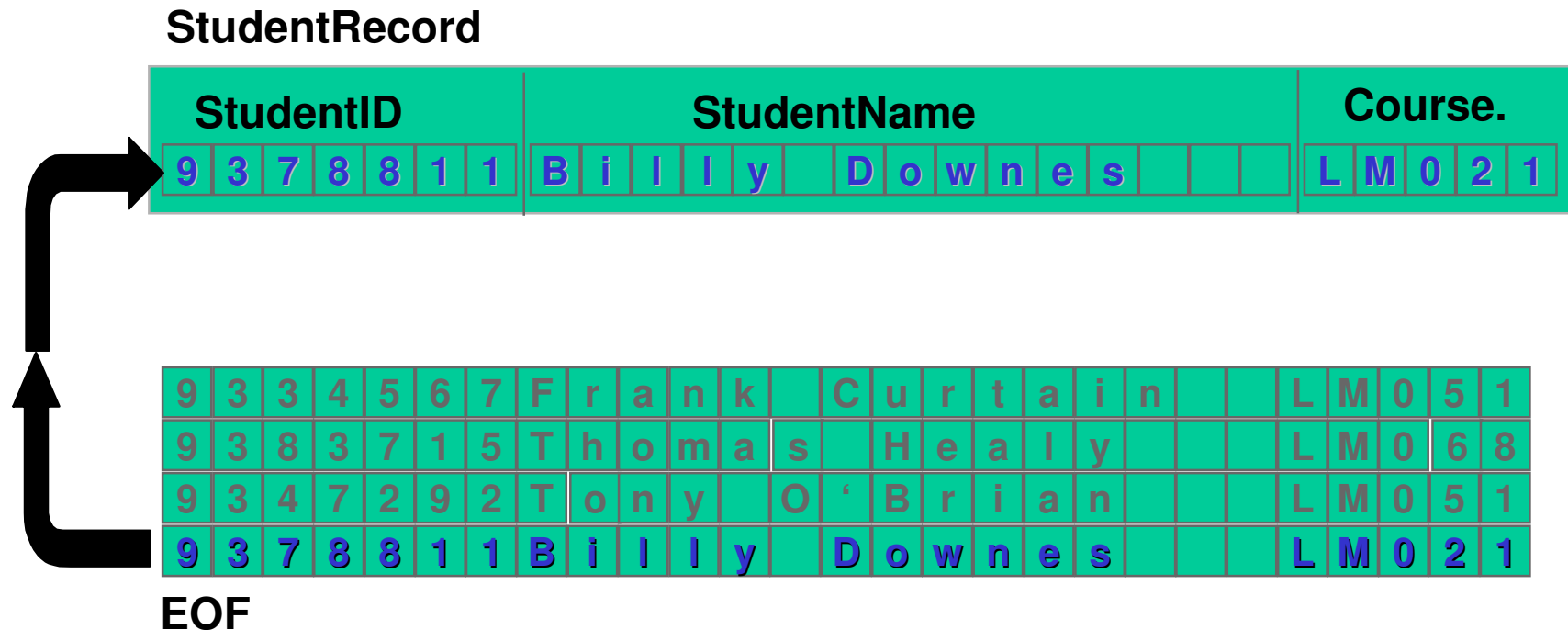
9334567	F	r	a	n	k		C	u	r	t	a	i	n		L	M	0	5	1
9383715	T	h	o	m	a	s		H	e	a	l	y		L	M	0	6	8	
9347292	T	o	n	y	O	'	B	r	i	a	n		L	M	0	5	1		
9378811	B	i	l	l	y		D	o	w	n	e	s		L	M	0	2	1	


EOF



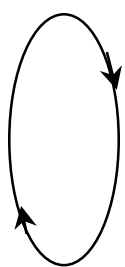
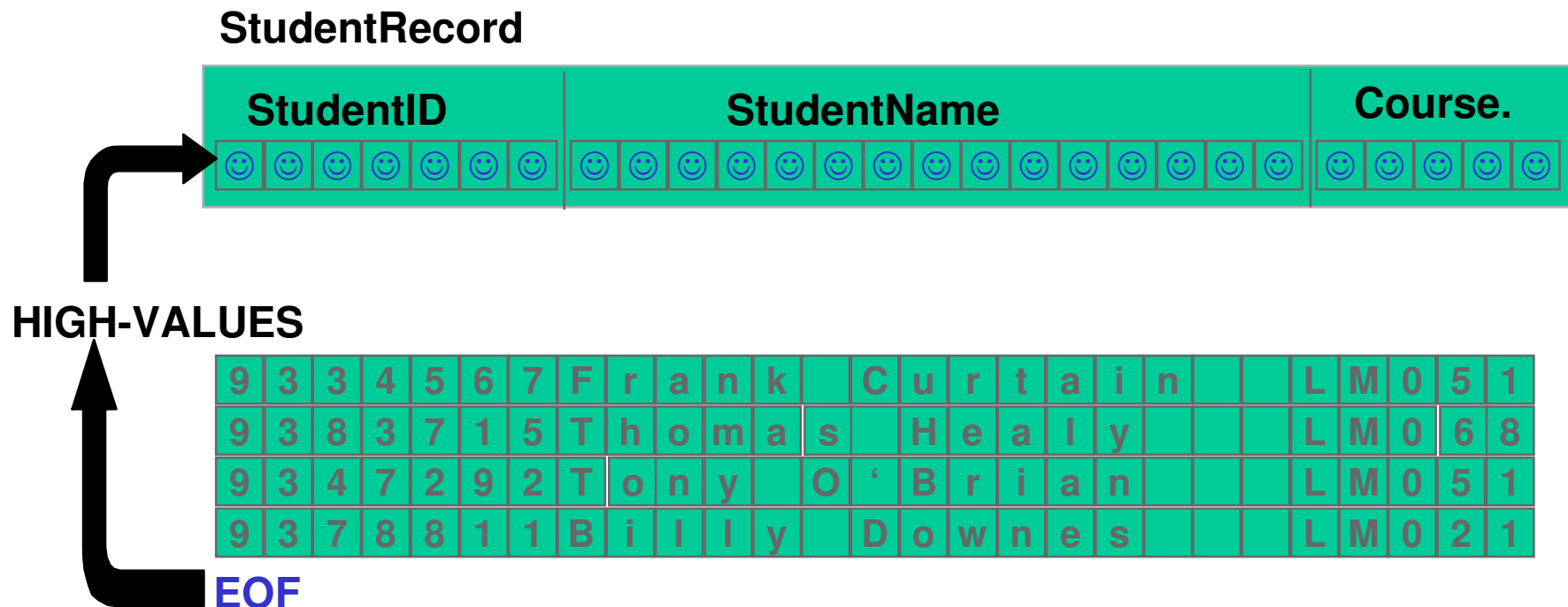
PERFORM UNTIL StudentRecord = HIGH-VALUES
READ StudentRecords
AT END MOVE HIGH-VALUES TO StudentRecord
END-READ
END-PERFORM.

How the READ works




PERFORM UNTIL StudentRecord = HIGH-VALUES
READ StudentRecords
AT END MOVE HIGH-VALUES TO StudentRecord
END-READ
END-PERFORM.

How the READ works



PERFORM UNTIL StudentRecord = HIGH-VALUES
READ StudentRecords
AT END MOVE HIGH-VALUES TO StudentRecord
END-READ
END-PERFORM.

How the WRITE works

```
OPEN OUTPUT StudentFile.  
MOVE "9334567Frank Curtain  LM051" TO StudentDetails.  
WRITE StudentDetails.  
MOVE "9383715Thomas Healy  LM068" TO StudentDetails.  
WRITE StudentDetails.  
CLOSE StudentFile.  
STOP RUN.
```

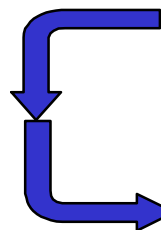
StudentRecord

StudentID	StudentName	Course.
9 3 3 4 5 6 7	F r a n k C u r t a i n	L M 0 5 1

Students.Dat

9 3 3 4 5 6 7 F r a n k C u r t a i n L M 0 5 1

EOF



How the WRITE works

```
OPEN OUTPUT StudentFile.  
MOVE "9334567Frank Curtain LM051" TO StudentDetails.  
WRITE StudentDetails.  
MOVE "9383715Thomas Healy LM068" TO StudentDetails.  
WRITE StudentDetails.  
CLOSE StudentFile.  
STOP RUN.
```

StudentRecord

StudentID	StudentName	Course.
9 3 8 3 7 1 5	T h o m a s H e a l y	L M 0 6 8

Students.Dat

9 3 3 4 5 6 7 F r a n k C u r t a i n L M 0 5 1
9 3 8 3 7 1 5 T h o m a s H e a l y L M 0 6 8

EOF

The syntax of the Read statement for a sequential disk file

```
READ file-name [RECORD]
    AT END
        imperative-statement-1 ...
[NOT AT END
    imperative-statement-2 ...]
```

Some typical Read statements for sequential disk files

```
READ CUSTMAST RECORD
    AT END
        MOVE "Y" TO CUSTMAST-EOF-SWITCH.

READ CUSTMAST
    AT END
        MOVE "Y" TO CUSTMAST-EOF-SWITCH
    NOT AT END
        ADD 1 TO RECORD-COUNT.
```

The syntax of the Write statement for a print file

WRITE record-name

AFTER ADVANCING $\left\{ \begin{array}{l} \text{PAGE} \\ \text{integer [LINE | LINES]} \\ \text{data-name [LINE | LINES]} \end{array} \right\}$

Some typical Write statements for print files

WRITE PRINT-AREA

AFTER ADVANCING PAGE.

WRITE PRINT-AREA

AFTER ADVANCING 1 LINE.

WRITE PRINT-AREA

AFTER ADVANCING SPACE-CONTROL LINES.

How to code Read and Write statements

- When the Read statement is executed for a sequential disk file, it reads the next record in sequence into the record description for the file.
- If there are no more records in the file, the At End clause of the Read statement is executed; otherwise, the Not At End clause is executed.
- When the Write statement is executed for a print file, one record is printed from the print area for the file.
- If the After Advancing clause is included on a Write statement, the paper is advanced the indicated number of lines.
- If the Page clause is included on a Write statement, the paper is advanced to the top of the next page.
- You code the file name in a Read statement and the record name in a Write statement.

The Rewrite statement for sequential files

```
REWRITE record-name [FROM data-name]  
[END-REWRITE]
```

The Close statement for sequential files

```
CLOSE file-name-1 ...
```

Description

- The Rewrite statement writes the record in the record area back to the file.
- The Rewrite statement can be used only for a record that already exists and only after executing a Read statement for that record.
- The From clause of the Rewrite statement allows you to write the record from a specified area in working storage.
- The Close statement closes the specified files.

Data Organization Alternatives

Sequential file organization

- In a file that has *sequential file organization*, the records are stored one after the other in consecutive disk locations. This type of file can also be called a *sequential file*.
- To retrieve records from a sequential file, you must read them sequentially by disk location. This type of record retrieval is referred to as *sequential access*.
- The records in a sequential file may be in ascending or descending sequence based on the values in one or more *key fields*, or *keys*.
- Often, the records in a sequential file aren't in key sequence, so they have to be sorted into sequence before they can be processed.
- To improve I/O efficiency, the records in sequential files are usually blocked with a large blocking factor.

A sequential employee file that's in sequence by employee number

Disk location	Employee number	Social security number	First name	Middle initial	Last name
1	00001	499-35-5079	Stanley	L	Abbott
2	00002	279-00-1210	William	J	Collins
3	00004	899-16-9235	Alice		Crawford
4	00005	703-47-5748	Paul	M	Collins
5	00008	558-12-6168	Marie	A	Littlejohn
6	00009	559-35-2479	E	R	Siebert
7	00010	334-96-8721	Constance	M	Harris
8	00012	213-64-9290	Thomas	T	Bluestone
9	00013	572-68-3100	Jean	B	Glenning
10	00015	498-27-6117	Ronald	W	Westbrook

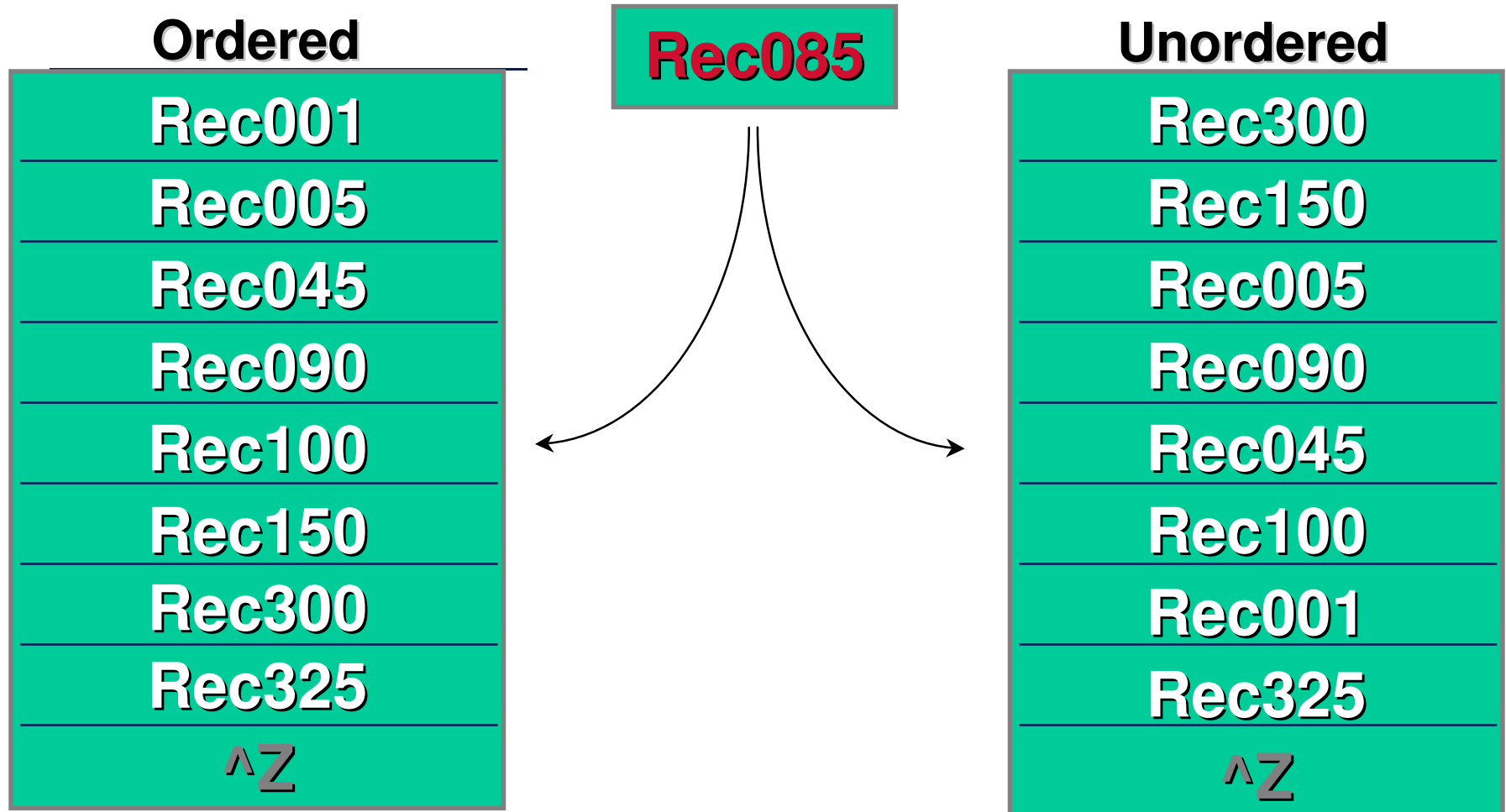
Entry-Sequenced Data Set (Sequential)

- Records in an ESDS are stored in the order in which they are written and are retrieved by addressed access.
- Records are loaded irrespective of their contents and their byte addresses cannot be changed.
- ESDS is also referred to as a **sequential VSAM** data set. This is because records in an ESDS are normally processed sequentially.
- ESDS is best suited for applications where most processing is done sequentially.
- **Sequential files may be Ordered or Unordered**

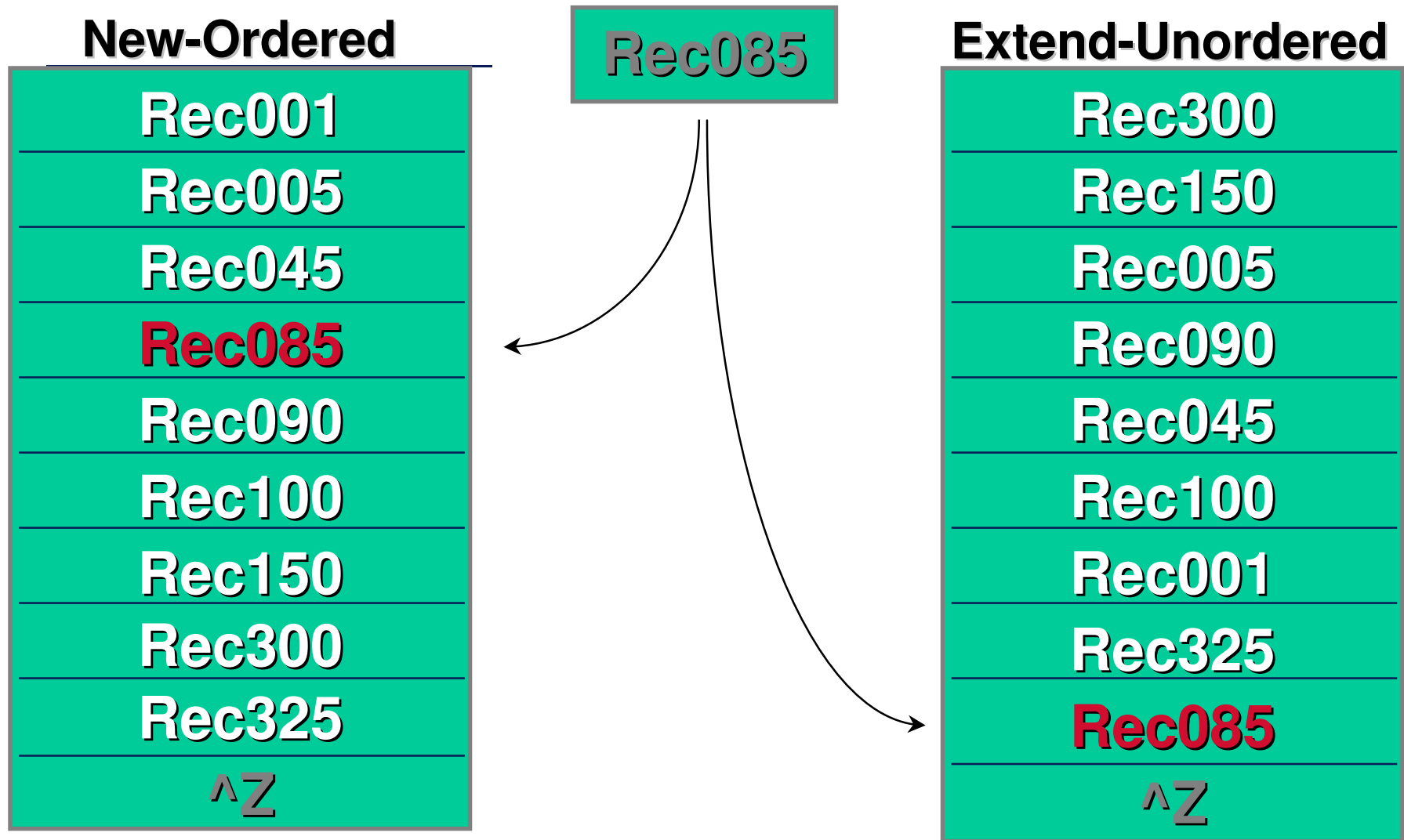
Problems with Sequential Files

- It is easy to **add** records to an unordered Sequential file, but:
 - Records in a Sequential file can not be deleted or updated “in situ” (unless the same size record).
 - The only way to update records in a Sequential File is to create a new file which contains the updated records.
 - The only way to delete Sequential file records is to create a new file which does not contain them.
 - Why?

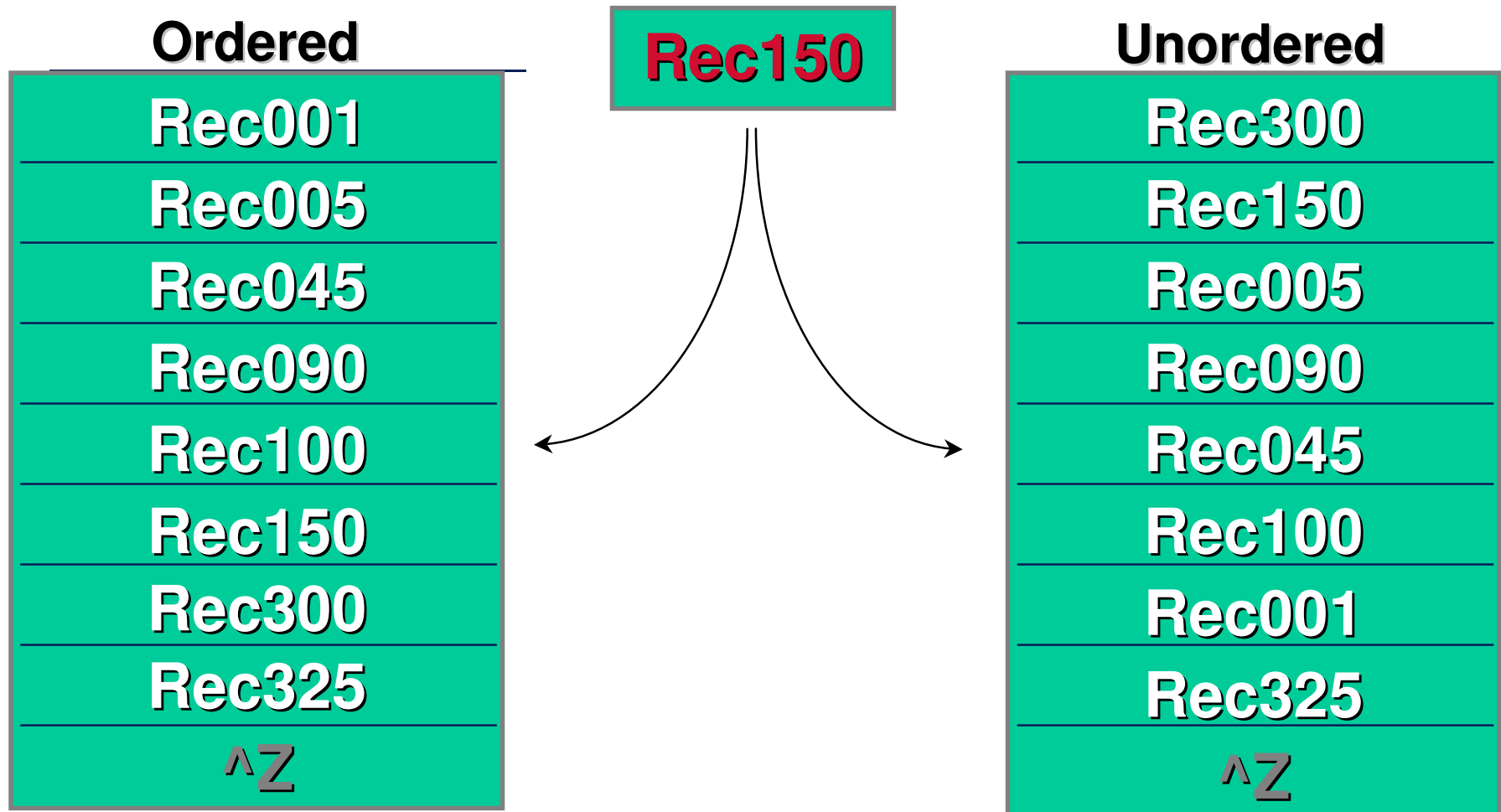
Sequential Files - Adding a Record



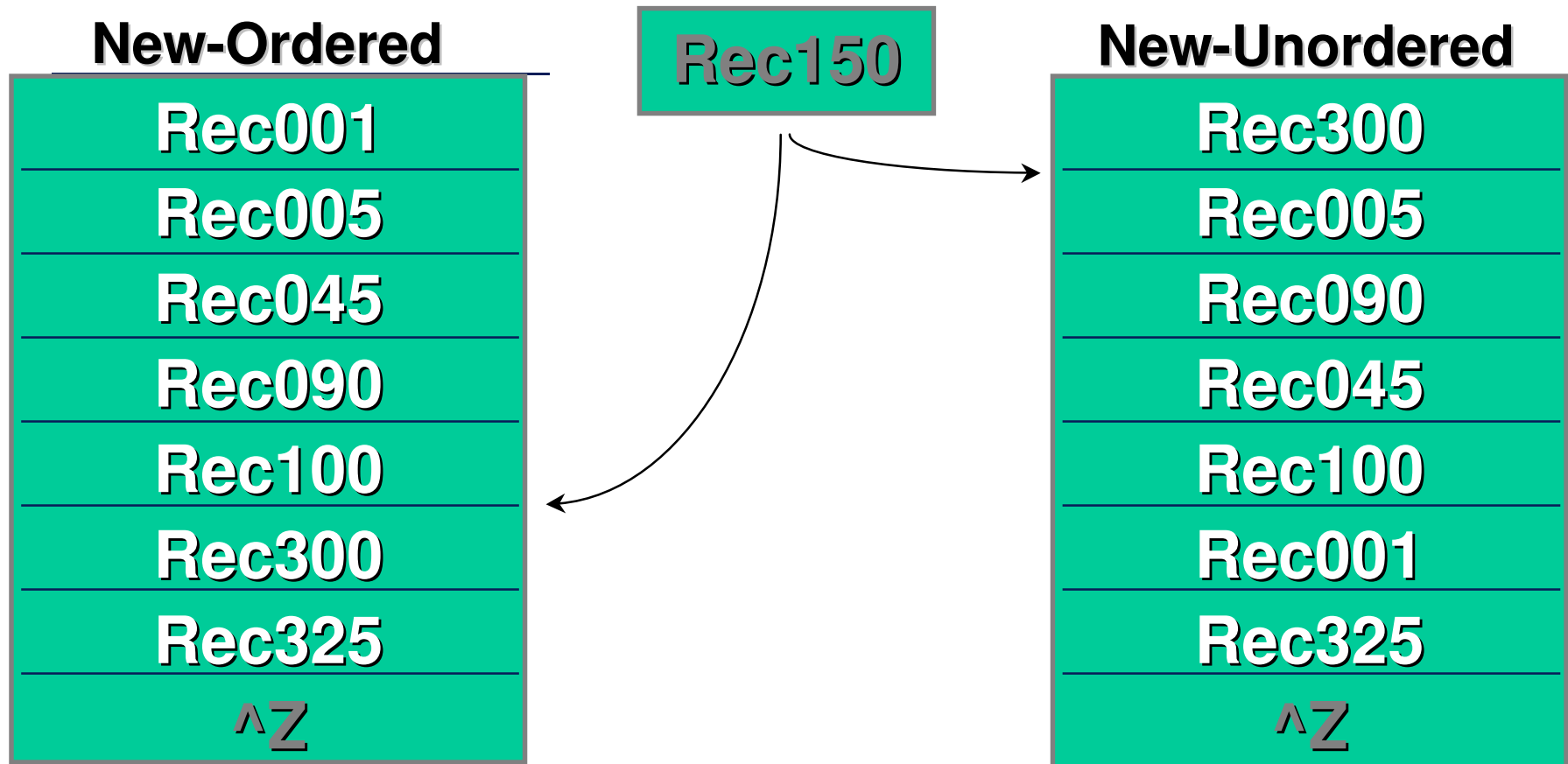
Sequential Files - Adding a Record



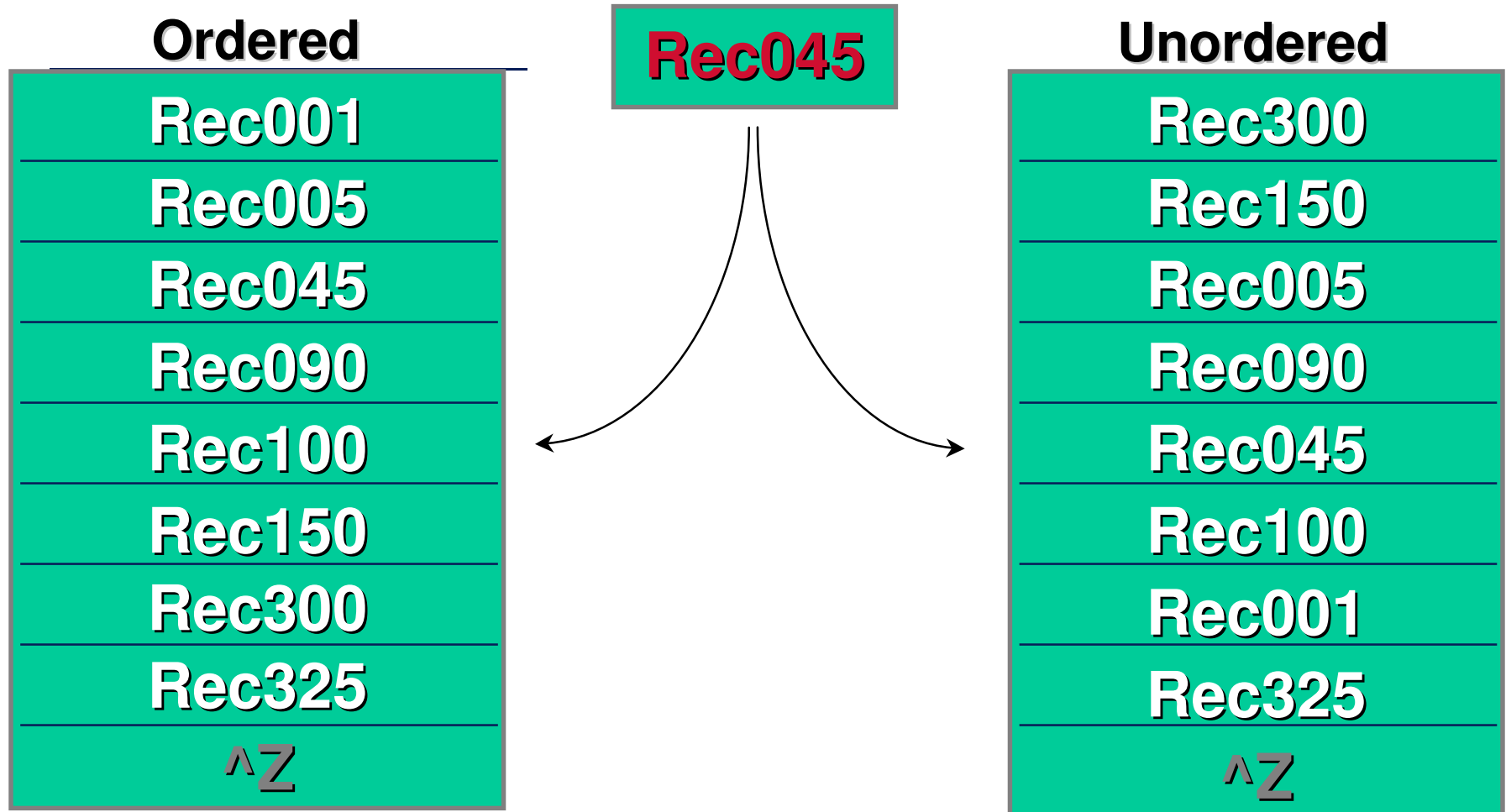
Sequential Files - Deleting a Record



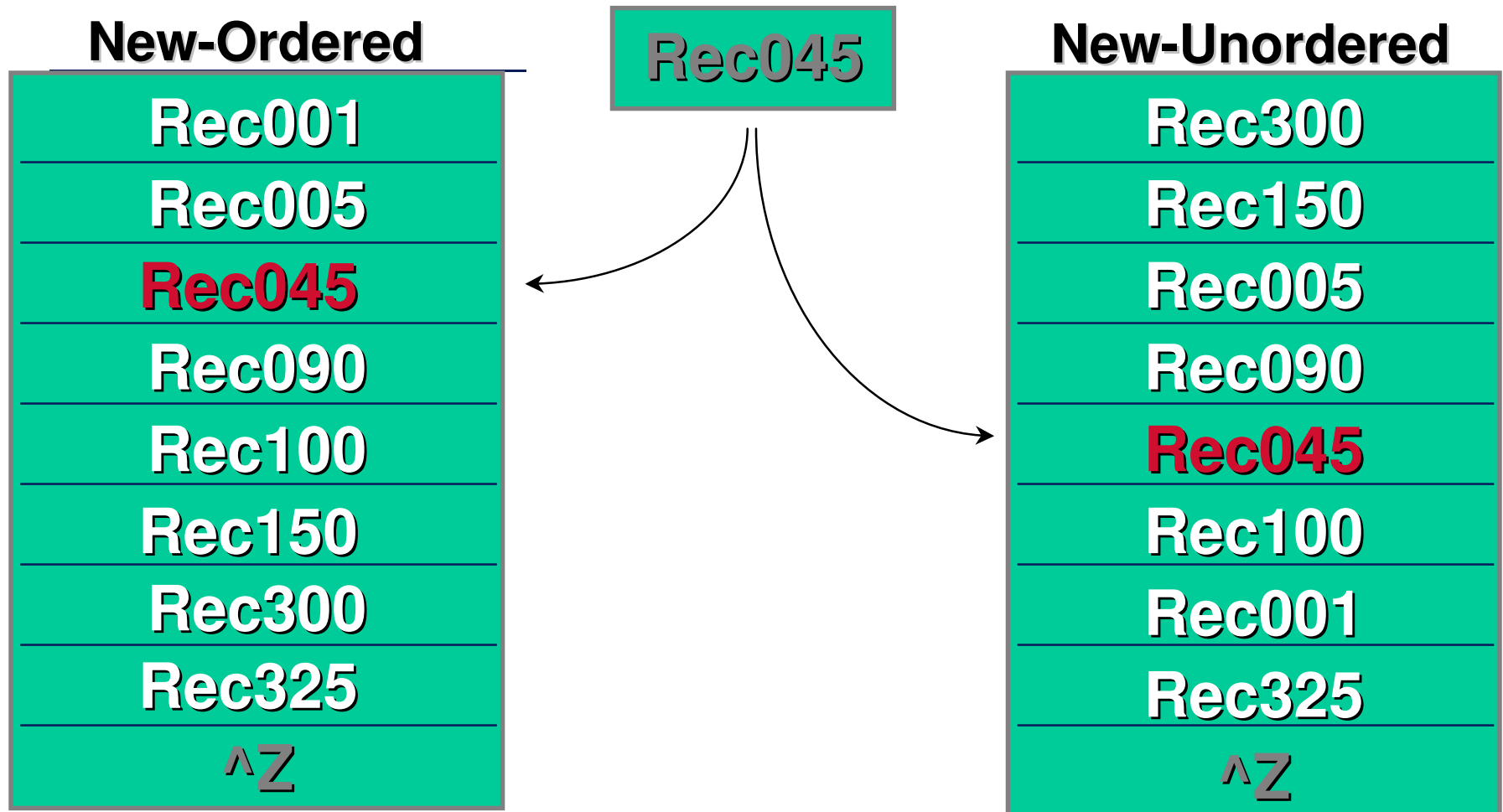
Sequential Files - Deleting a Record



Sequential Files - Amending a Record



Sequential Files - Amending a Record

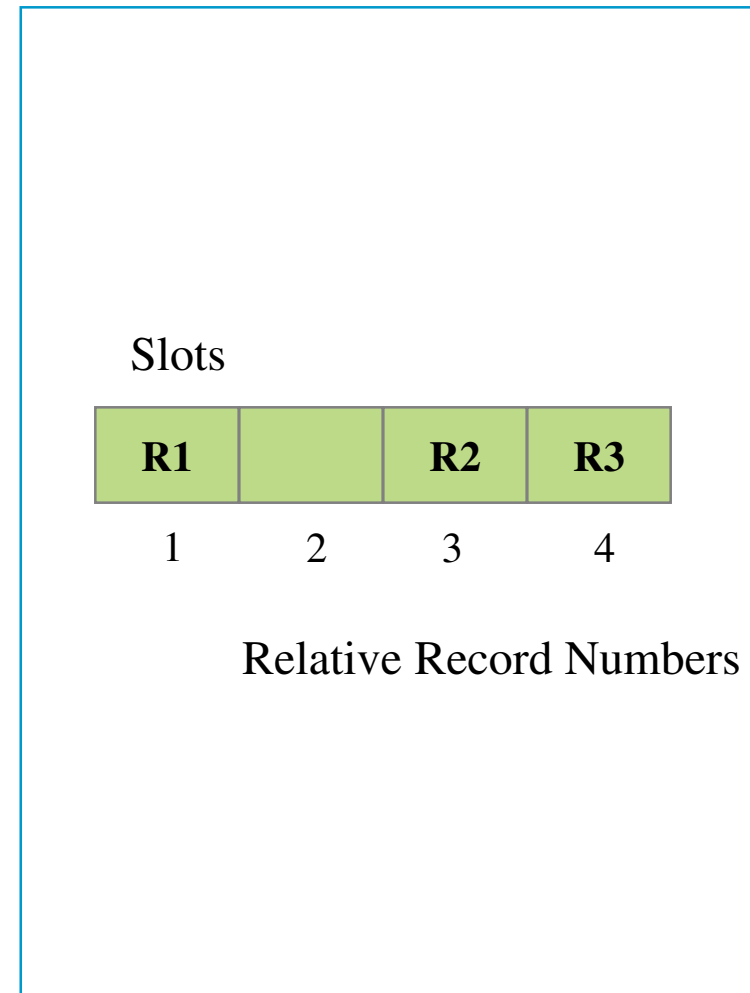


An employee file with relative organization

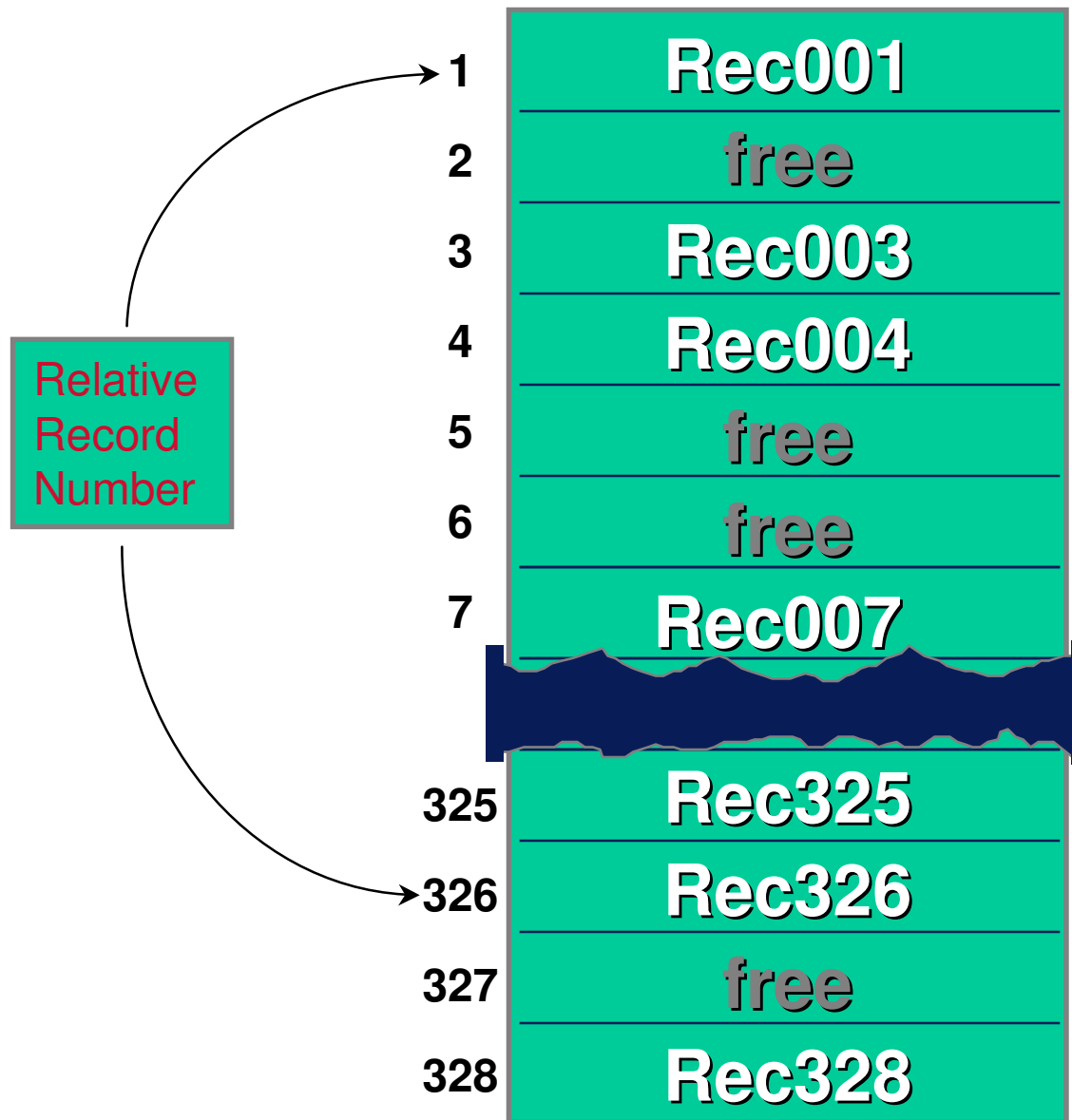
Relative record number	Employee number	Social security number	First name	Middle initial	Last name
1	00001	499-35-5079	Stanley	L	Abbott
2	00002	279-00-1210	William	J	Collins
3					
4	00004	899-16-9235	Alice		Crawford
5	00005	703-47-5748	Paul	M	Collins
6					
7					
8	00008	558-12-6168	Marie	A	Littlejohn
9	00009	559-35-2479	E	R	Siebert
10	00010	334-96-8721	Constance	M	Harris
11					
12	00012	213-64-9290	Thomas	T	Bluestone
13	00013	572-68-3100	Jean	B	Glenning
14					
15	00015	498-27-6117	Ronald	W	Westbrook

Relative Record Data Set

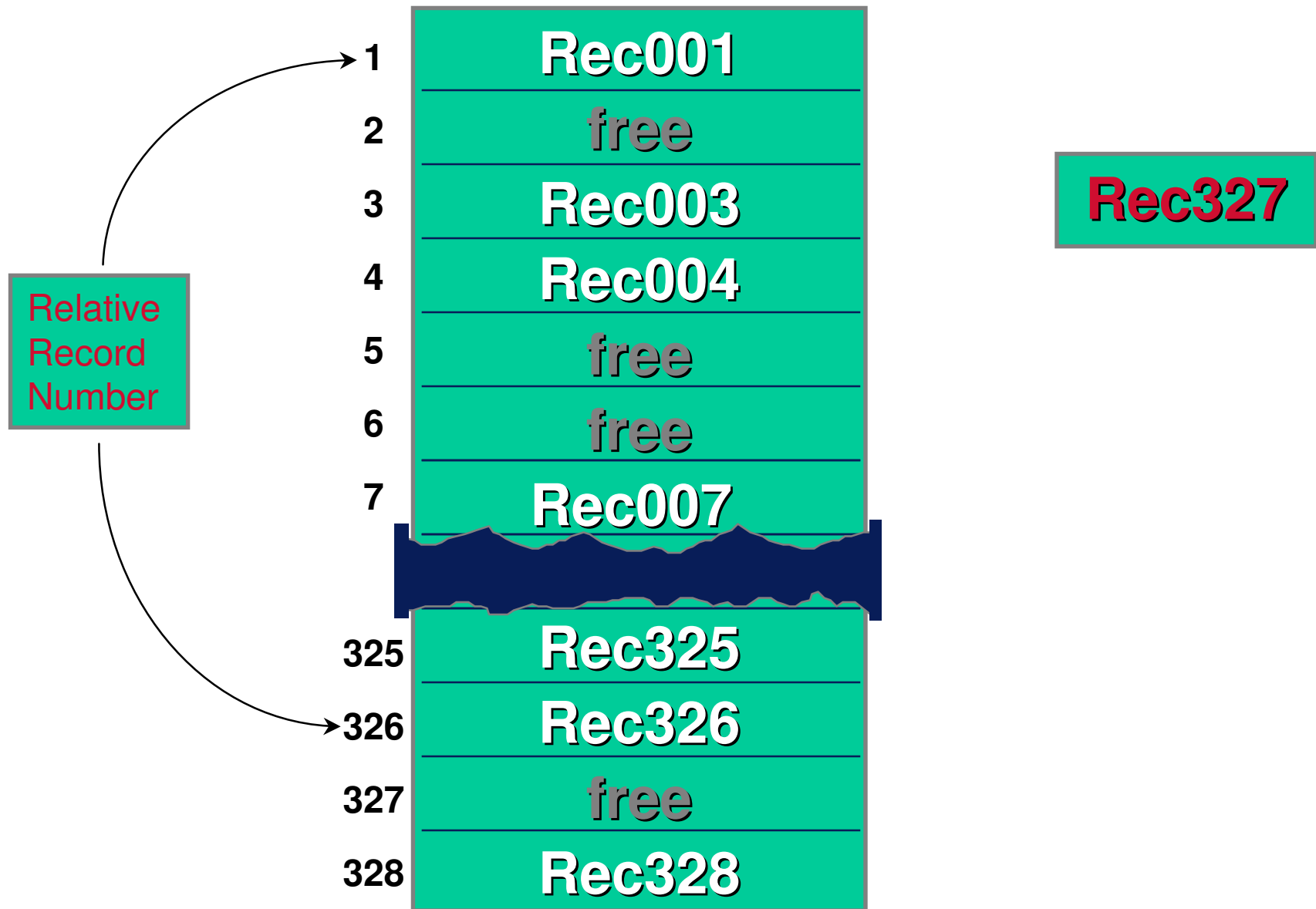
- Records in an RRDS are loaded into fixed-length or variable length slots. These records are represented by the Relative Record Numbers (RRNs) of their slots.
- A processing program uses RRN to provide random access to records.
- The records in an RRDS can also be accessed sequentially in its RRN order. It is also possible to convert key values into RRNs.



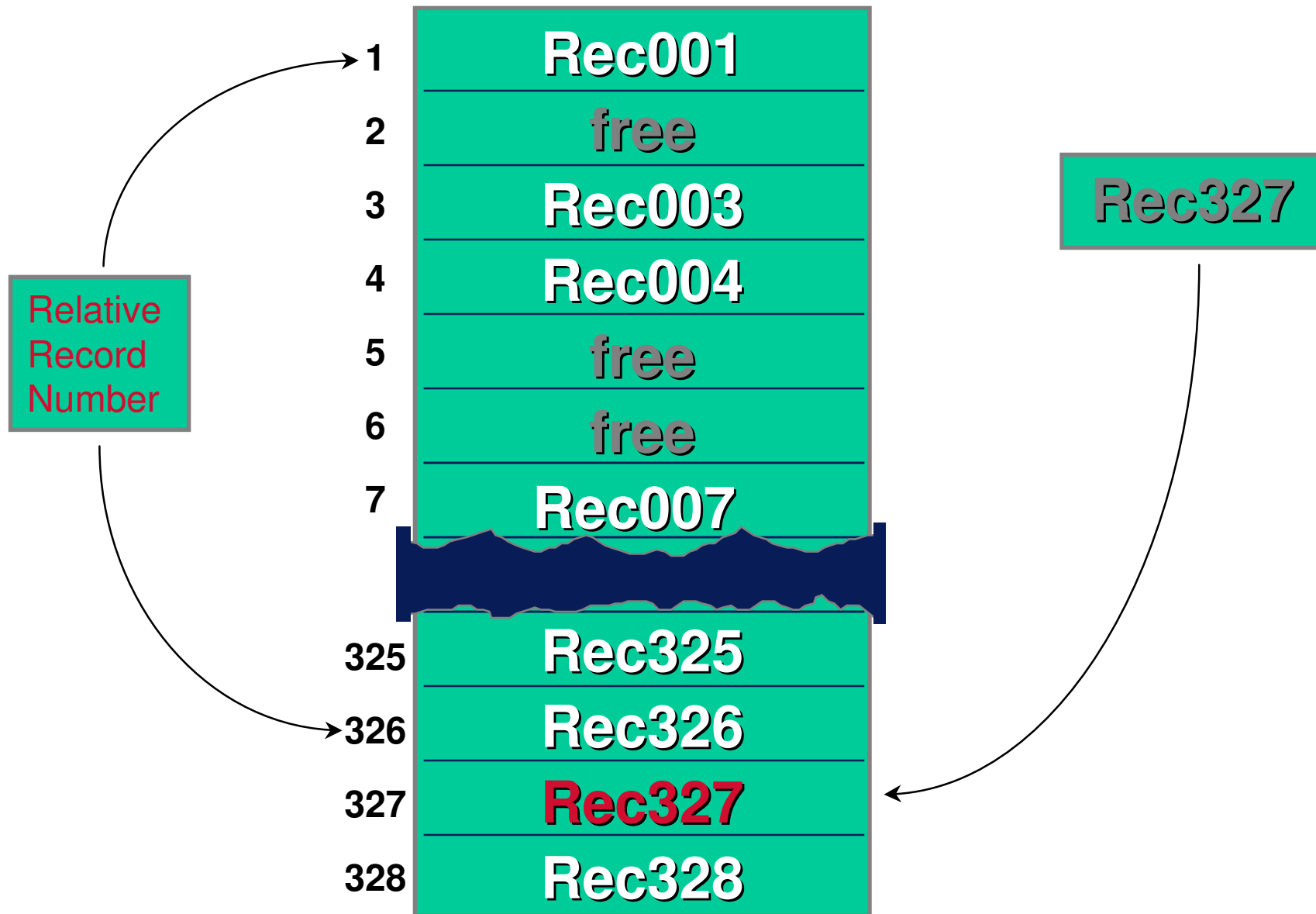
Relative Files - Organization



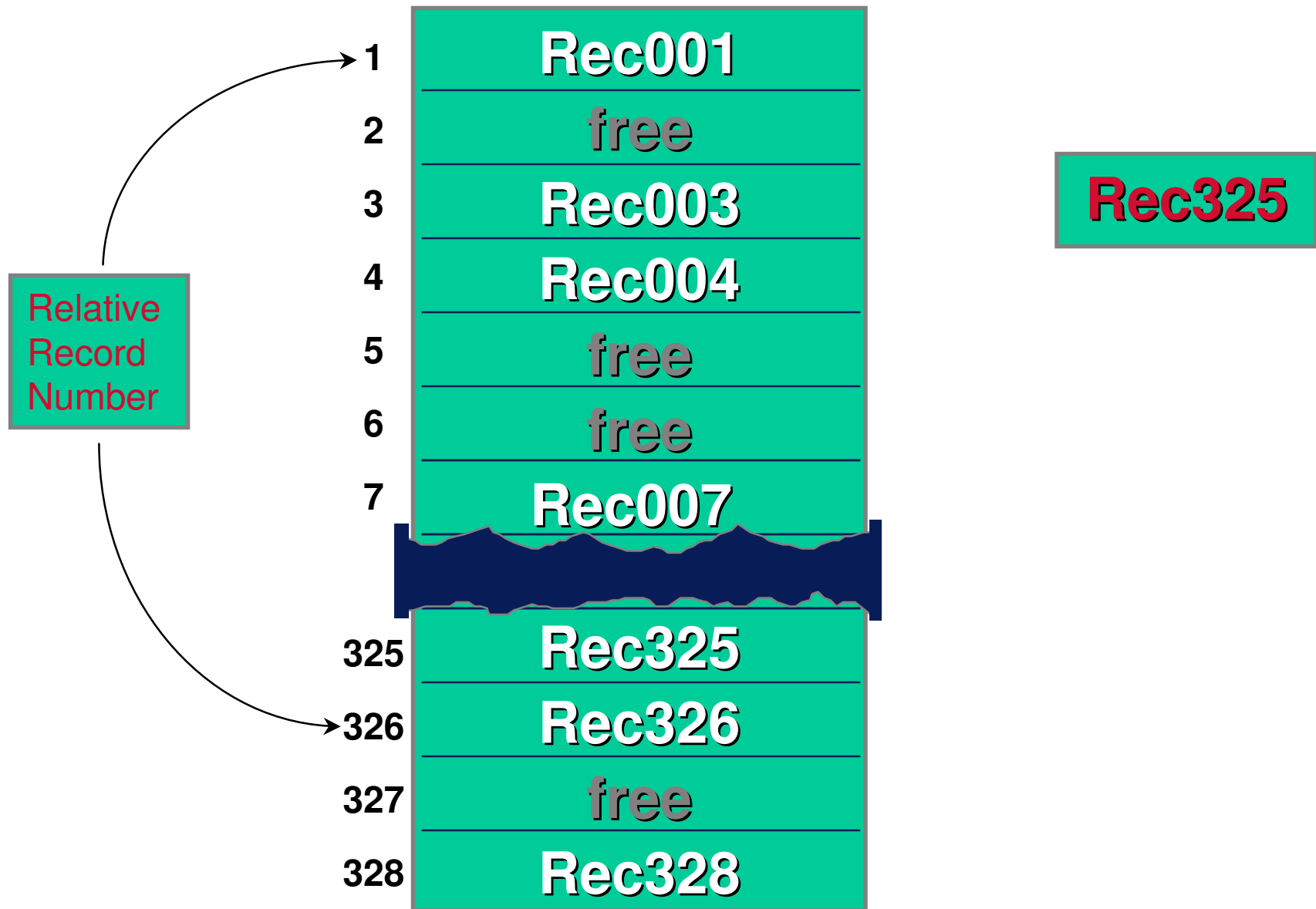
Relative Files - Adding a Record



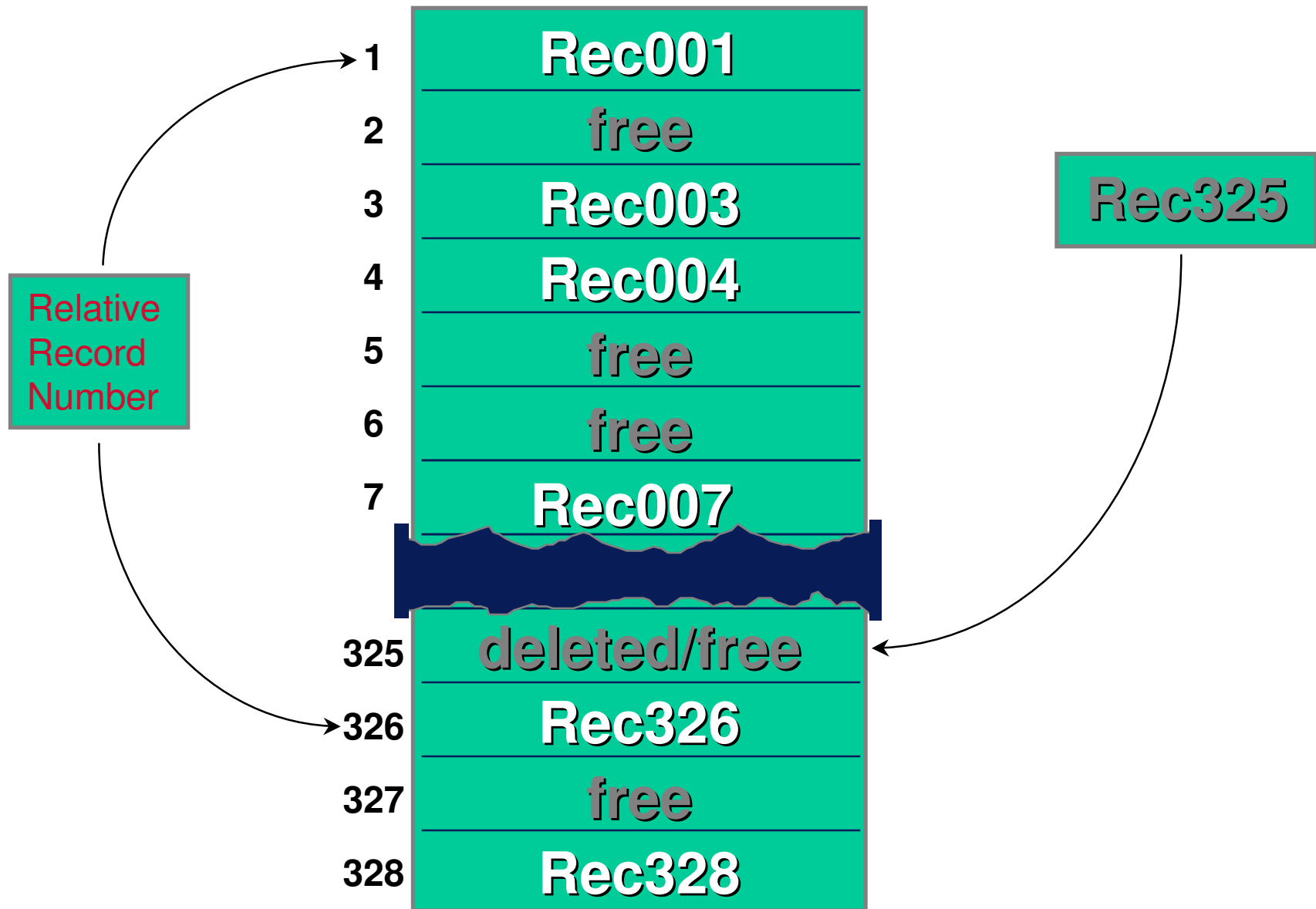
Relative Files - Adding a Record



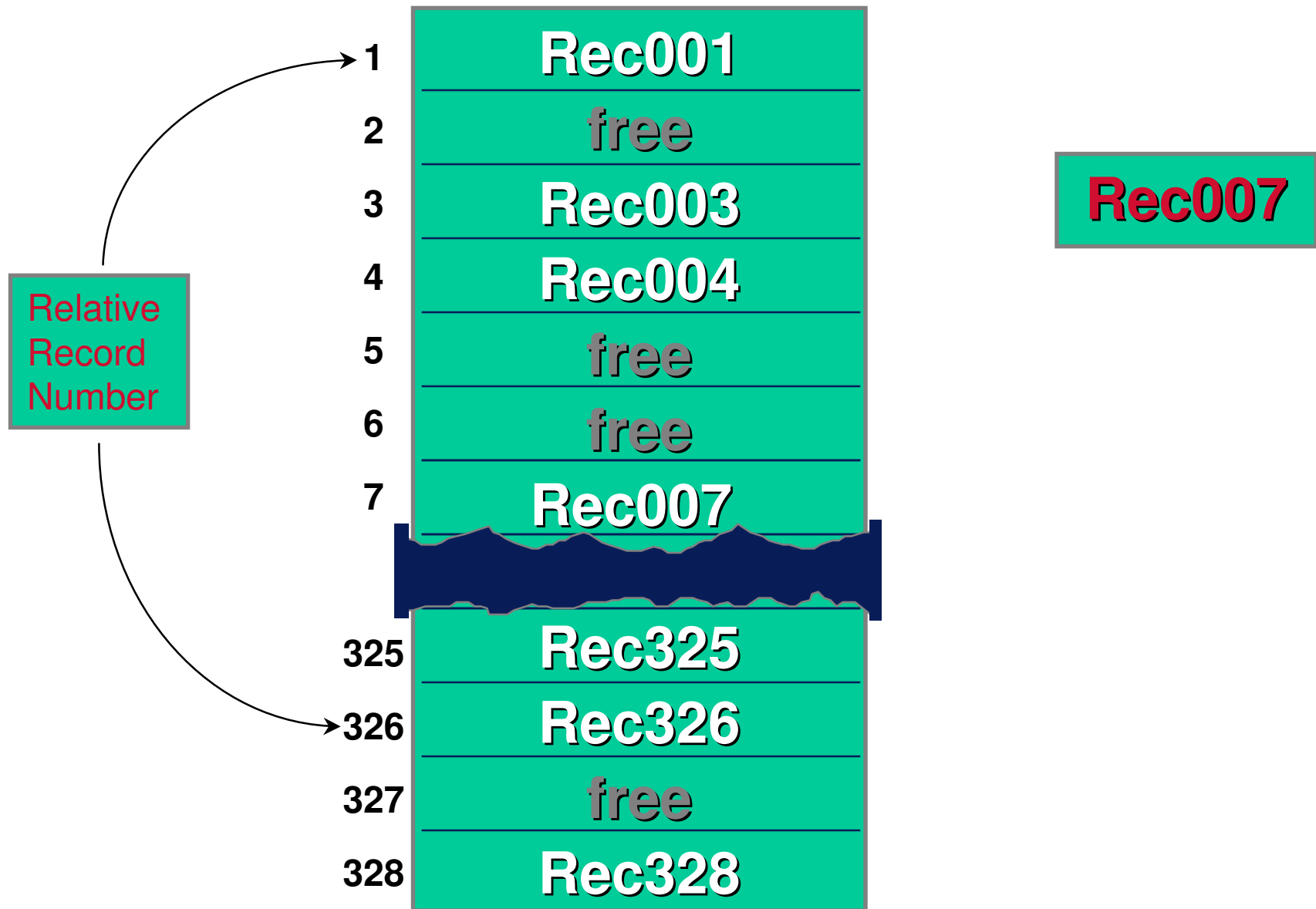
Relative Files - Deleting a Record



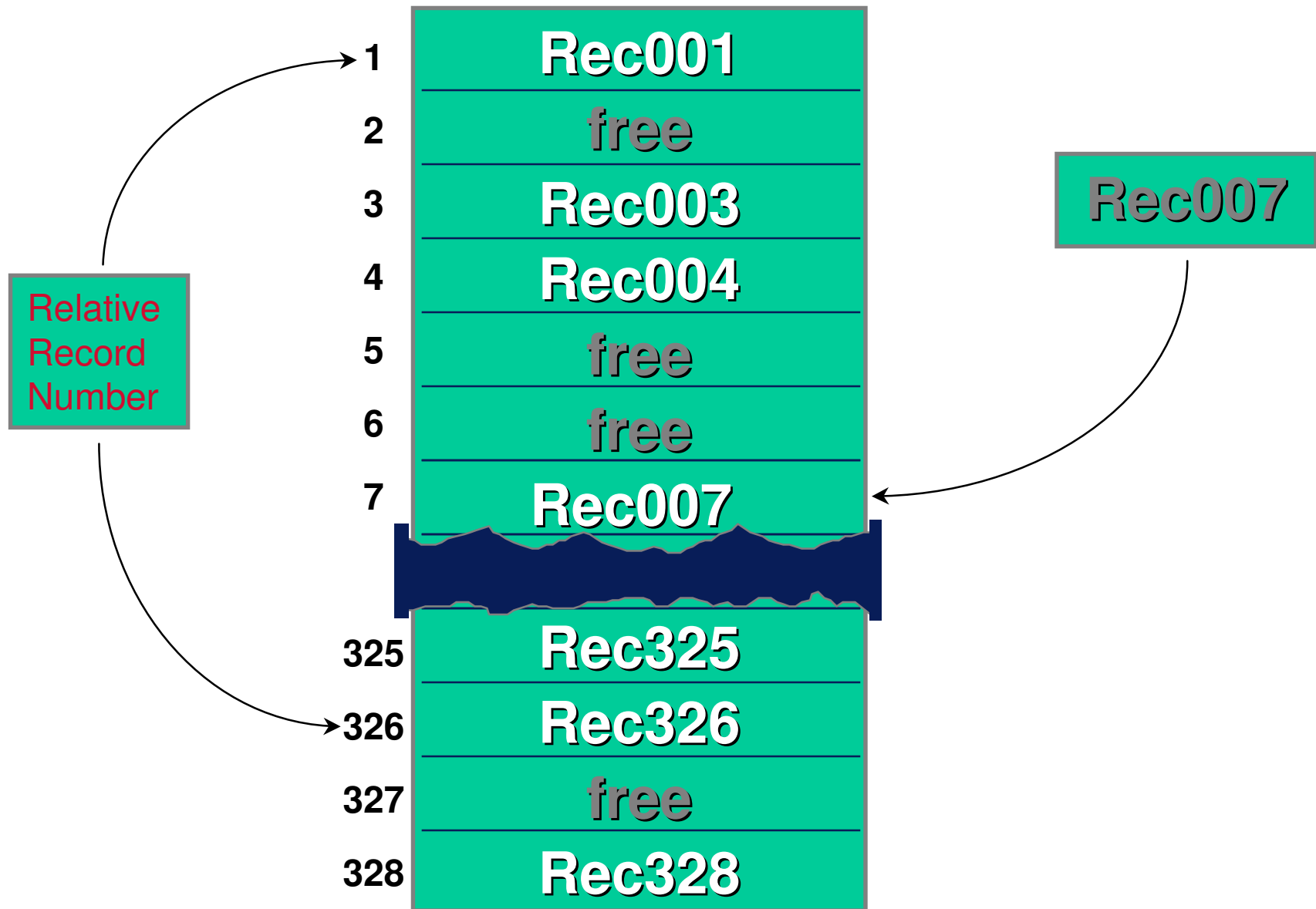
Relative Files - Deleting a Record



Relative Files - Amending a Record



Relative Files - Amending a Record



Relative file organization

- A file that has *relative file organization* consists of *record areas* that can contain one record each. This type of file can be called a *relative file*.
- Each record area is identified by a *relative record number* that indicates its relative position in the file.
- When the records in a relative file are accessed sequentially, the areas that don't contain records are skipped.
- To access a record randomly, a program must specify the relative record number of the area that contains the record.
- The relative record number for a record is usually derived from a key field. Usually, the key value has to be converted to a relative record number by using a *randomizing routine*.
- If two or more records randomize to the same relative record number, additional routines must be used to put each record with a duplicate record number in an available area.

Key-Sequenced Data Set (Indexed)

- Records in a KSDS are stored in key sequence and are **controlled by an index**. The key field of records determine the order in which records are stored.
- In a KSDS, records can be **processed both sequentially and randomly** using their key field values.
- The advantages of KSDS are:
 - Sequential processing is useful for retrieving records in the sorted form
 - Random or direct processing of records is useful in on-line applications

An employee file that's indexed by employee number

Index component

Employee number	Disk location
00001	1
00002	2
00004	3
00005	4
00008	5
00009	6
00010	10
00012	7
00013	8
00015	9

Data component

Employee number	Social security number	First name	Middle initial	Last name
00001	499-35-5079	Stanley	L	Abbott
00002	279-00-1210	William	J	Collins
00004	899-16-9235	Alice		Crawford
00005	703-47-5748	Paul	M	Collins
00008	558-12-6168	Marie	A	Littlejohn
00009	559-35-2479	E	R	Siebert
00012	213-64-9290	Thomas	T	Bluestone
00013	572-68-3100	Jean	B	Glenning
00015	498-27-6117	Ronald	W	Westbrook
00010	334-96-8721	Constance	M	Harris



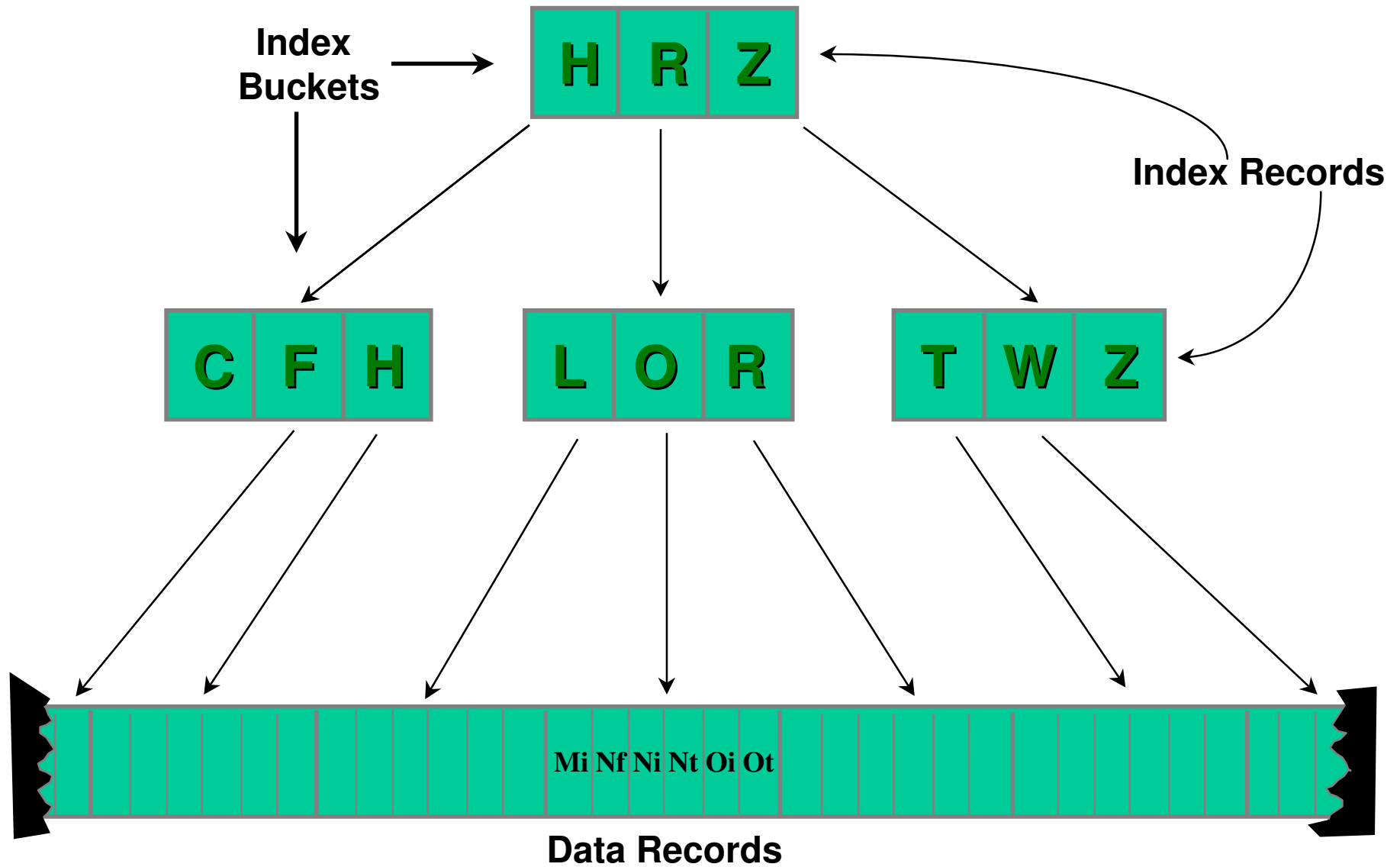
An employee file with unique alternate keys

Alternate index

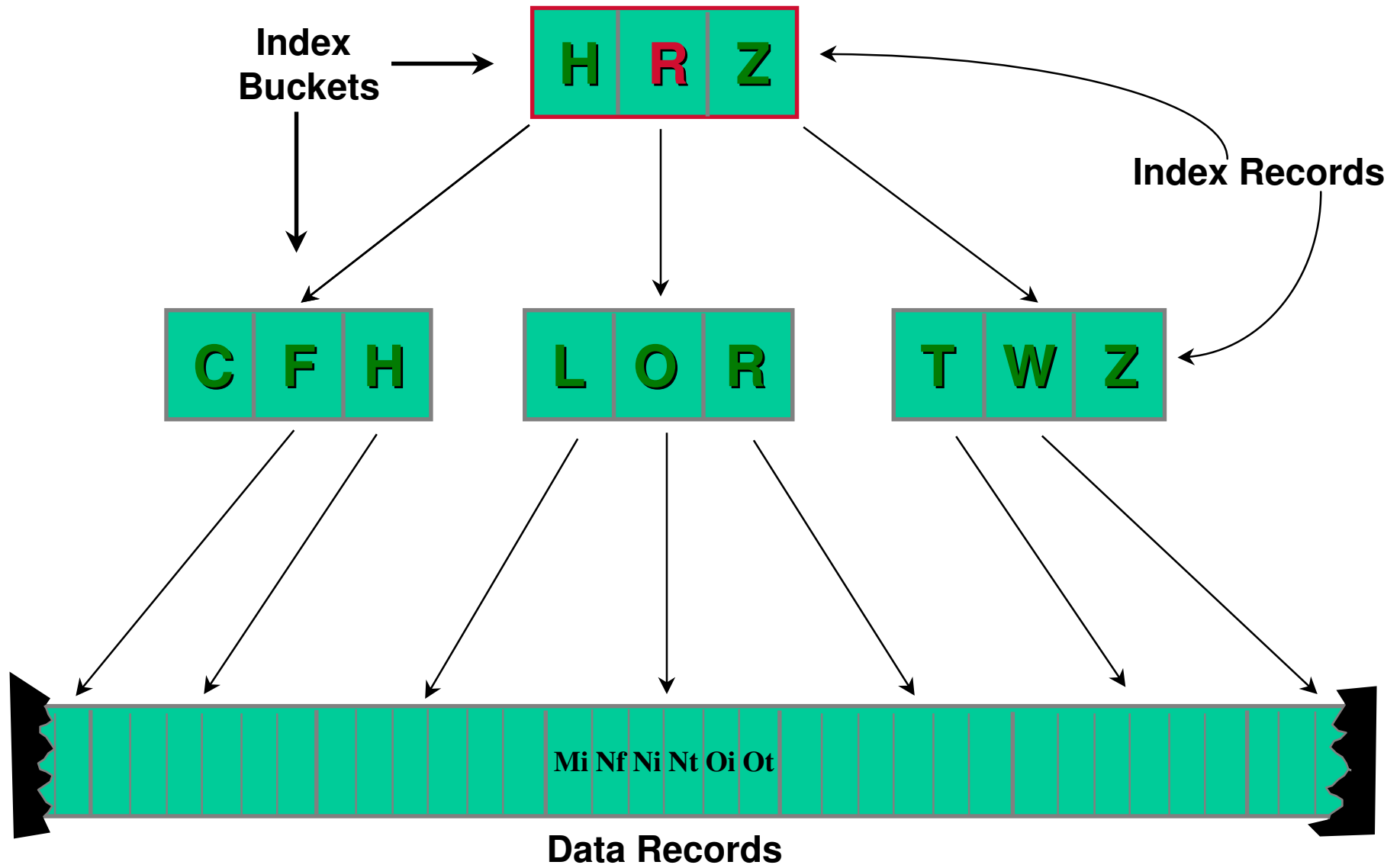
Data component

Social security number	Disk location	Disk location	Employee number	Social security number	First name	Middle initial	Last name
213-64-9290	7	1	00001	499-35-5079	Stanley	L	Abbott
279-00-1210	2	2	00002	279-00-1210	William	J	Collins
334-96-8721	10	3	00004	899-16-9235	Alice		Crawford
498-27-6117	9	4	00005	703-47-5748	Paul	M	Collins
499-35-5079	1	5	00008	558-12-6168	Marie	A	Littlejohn
558-12-6168	5	6	00009	559-35-2479	E	R	Siebert
559-35-2479	6	7	00012	213-64-9290	Thomas	T	Bluestone
572-68-3100	8	8	00013	572-68-3100	Jean	B	Glenning
703-47-5748	4	9	00015	498-27-6117	Ronald	W	Westbrook
899-16-9235	3	10	00010	334-96-8721	Constance	M	Harris

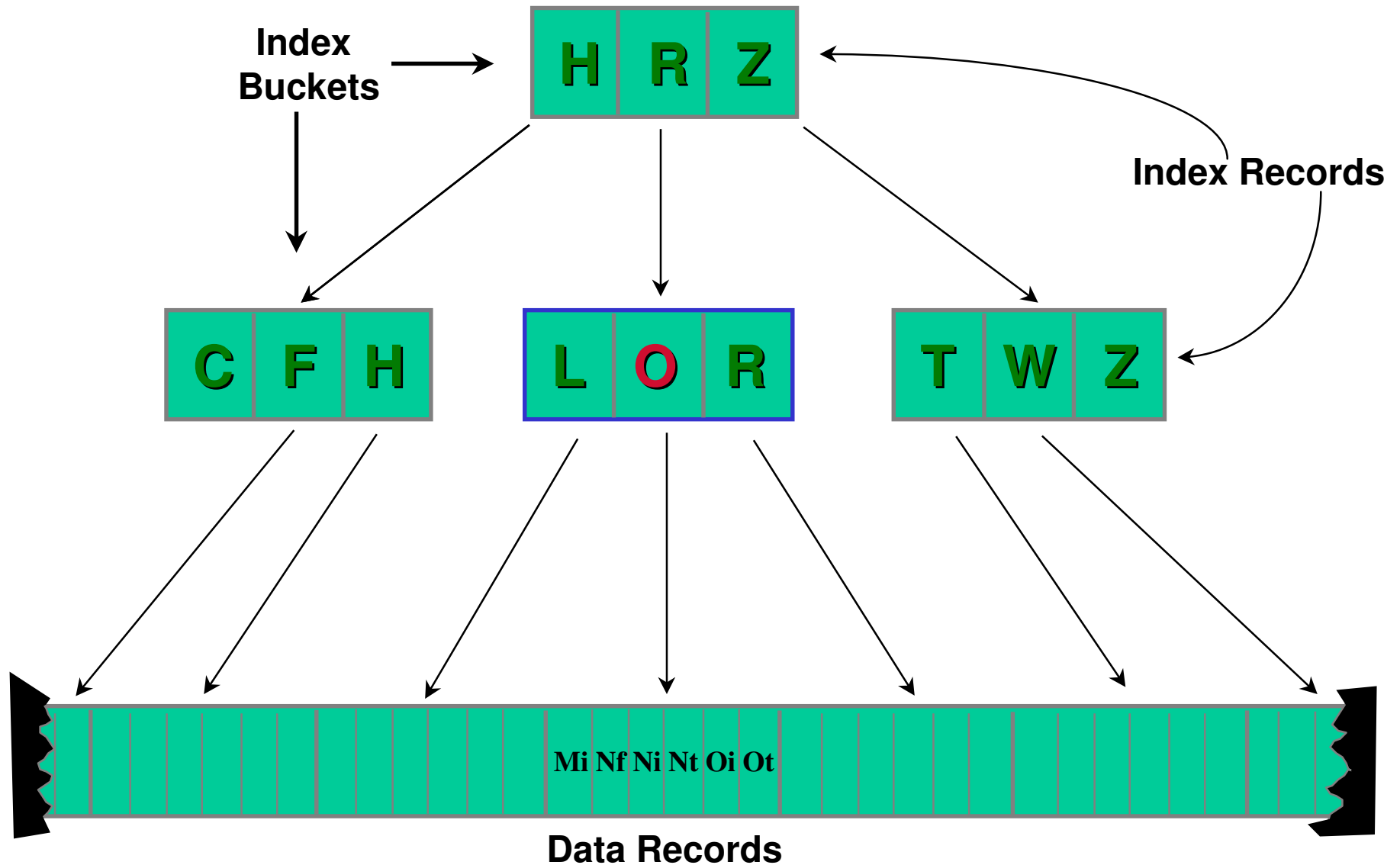
Indexed Files - Organization



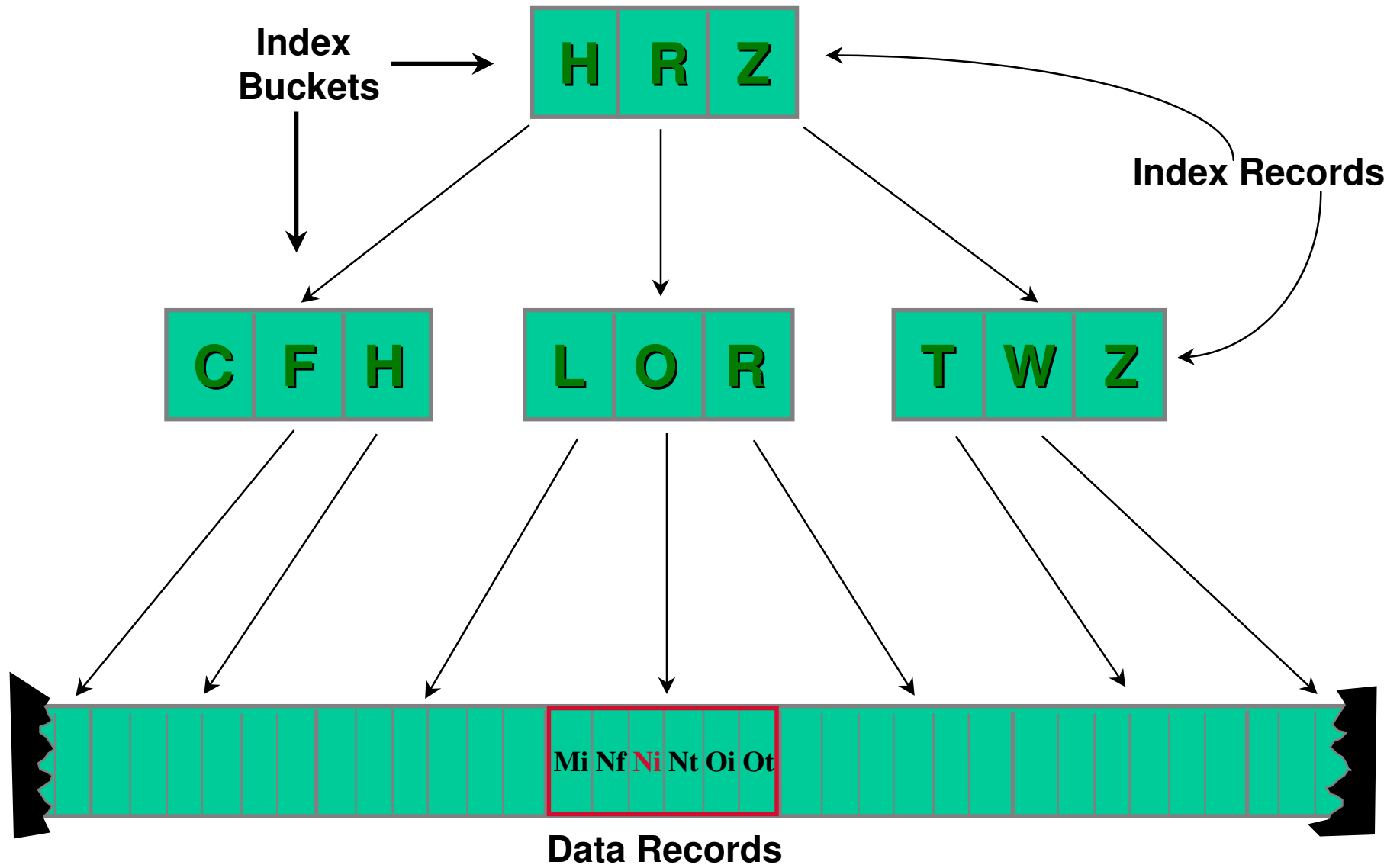
Indexed Files - Reading Record Ni



Indexed Files - Reading Record Ni



Indexed Files - Reading Record N_i



Indexed file organization

- A file that has *indexed file organization* consists of two parts: an *index component* and a *data component*. This type of file can be called an *indexed file*.
- Within the index component, each entry contains a key field value and the location of the corresponding record in the data component.
- An indexed file must have at least one index. The key for this index is called the *primary key*, and the index is called the *primary index*.
- Each key in the primary index must be *unique*. That means that there are no *duplicate keys* in the file.
- In the primary index, the primary keys are in sequence. That way, the index can be used to read the records in the data component sequentially. This is referred to as *sequential access*.

Indexed file organization (continued)

- The index can also be used to access a record directly by getting its disk location from the index component. This is referred to as *direct*, or *random*, *access*.
- When an indexed file is created, the records are written in their primary key sequence in the data component. That reduces actuator movement during sequential access.
- When records are added to the file, they may not be added in their sequential locations in the data component. That reduces the efficiency of sequential access.
- To improve I/O efficiency, the records in indexed files are often blocked.

Reviewing Data Sets: When to use what - an application perspective

Interactive programs

- In an *inquiry program*, the computer user enters the key of a record. Then, the program finds that record and displays some or all of its data, but it doesn't allow changes to that data.
- In a *maintenance program*, the computer user enters *transactions* to add, delete, or change master records.
- As a maintenance transaction is entered, the program *edits* it for validity. If the data is valid, the program maintains the master. If not, the user is asked to correct the data.
- In an *update program*, the computer user enters transactions like customer orders or course grades. For each valid transaction, the program updates all of the related master records, which can be in more than one file.
- **Most interactive programs use indexed master files so they can access the records that the user wants to work with on a random basis.**

Batch updating of an indexed file

- An interactive *edit program* gets transactions from a computer user, edits them, and saves the valid transactions in a sequential file.
- A *random update program* reads the valid transaction records, then updates the records in the master file on a random basis.
- Usually, a random update program also writes a transaction to a file of error transactions if a master record can't be found with the key of the transaction record.
- A *random maintenance program* is like an update program, except the transactions are for records to be added, deleted, or changed.
- A master file can also be updated or maintained sequentially. Often, this is more efficient than random processing because the I/O time is reduced.
- To improve the efficiency of indexed file processing, an indexed file is periodically recreated.

Batch updating of a sequential file

- An interactive edit program gets the transactions from a computer user and saves the valid transactions in a sequential disk file.
- A *sort program* sorts the transactions into the key sequence of the master file that's going to be updated. A sort program is a utility program that's provided by the operating system.
- A *sequential update program* reads the records in the valid transaction file and the records in the *old master file*. It then updates the master records and writes all of the old records to a *new master file*.